An Introduction to Code Composer Studio
ECE 443/643 Communications Lab #4: NDSU : Fall, 2006

Roger Green, Brian Chapman, David Farden

Abstract

Code Composer Studio (CCS) is an integrated development environment used with Digital Signal Processors (DSP) manufactured by Texas Instruments (TI). This laboratory introduces CCS as well as a powerful TI DSP: the TMS320C6713. The 320C6x family of DSP's are capable of over a billion operations per second, and come equipped with a wide range of on-chip peripheral devices. DSPs are found in a broad range of commercial devices such as cellular phones.

1 Prelab

Be sure to read the entire laboratory handout prior to attending lab. That's it!

2 Introduction

Spectral inversion is an application guaranteed to impress at least one of your friends (or, lacking friends, you can impress yourself). More importantly, it requires very little effort; a fact your friends need not know. Here is the basic idea. Sample a signal $x(t)$ using an analog-to-digital converter (ADC) to obtain a discrete-time signal $x[n]$. Next, create an output signal $y[n]$ according to $y[n] = (-1)^n x[n]$. Finally, send $y[n]$ through a digital-to-analog converter (DAC) to obtain the continuous-time output signal $y(t)$.

Although spectral inversion is pretty neat all by itself, it serves as a good beginning example for learning about CCS and TI DSP's. All the elements necessary to perform more sophisticated signal processing are included: sample acquisition, mathematical relation between input and output, and generation of an analog output.

3 Creating a CCS Project

CCS is a sophisticated product. It is impossible to learn everything about this package in a single lab. Rather, a basic procedure is followed that serves as a starting point for future projects.

- Create a new folder called `lab4` and place it on your Z-drive under a directory called `myprojects`. Your TA can tell you where to find the `common_code` directory – copy this into the same `myprojects` directory. The `myprojects` directory includes various required files:
  1. `DSK_Support.c,` functions to help initialize the DSK.
  2. `lnk6713.cmd,` linker command file to assign memory locations.
  3. `main.c,` main code that initializes DSK and waits for interrupts.
  4. `vectors.asm,` establishes vectors for interrupt service routines.
- Launch Code Composer Studio.
- From the CCS **project** menu, select **new**, type `lab4` as the project name, and then specify the directory as the `lab4` directory on your Z-drive. This process creates a new file `lab4.pjt` that stores information pertinent to your project. The project structure is shown in a CCS window. Expand your project and look at how CCS categorizes project files. At this point, no files are included in the project.
- From the CCS **project** menu, select the **add files to project** option. Add the four files listed above.

- From the CCS **project** menu, select the **add files to project** option. Add the two files `ISRs.c` and `StartUp.c`; again, your TA can help you locate these files. The file `ISRs.c` contains the interrupt service routines that are called to input or output samples to the codec, and this is where you will need to write the majority of your code. The file `StartUp.c` is called once by `main.c`, and is used for one-time configurations or initializations.
- Next, select **project** and then **scan all dependencies**. This will add the header (and other dependent) files to the project. Identify the files that were added to the project. Double-click to view the various files and see how they are constructed.

Now that the project is created, CCS needs to be told how to compile files. First, the **compiler** tab:

- Select **project** and the **build options**. Under the **basic** category, ensure that the **target version** is selected as **C671x**, that **generate debug info** is selected as **full symbolic debug**, that **opt speed vs size** is selected as **speed most critical**, and that no optimization levels are chosen.
- Under the **feedback** category, select **interlisting** to **Opt/C and ASM**. This will allow interlisting between C-code and generated assembly code.
- Under the **files** category, make sure the **obj directory** is simply your project directory. CCS sometimes creates a debug subdirectory, which is not necessary.
- Under the **assembly** category, click the **keep generated asm files** option. In this way, you can view assembly code generated by CCS.

The result is that the compiler options should read something like
`-g -k -s -fr"$(Proj_dir)\Debug" -d"_DEBUG" -mv6710`.
Next, the linker tab should be selected:

- Ensure the **output filename** reads `lab4.out`.

Adjust the other options so the linker script reads something like `-q -c -m".\Debug\lab4.map" -o".\Debug\lab4.out" -w -x`. Spend a little time investigating the various compiler and linker options.

4 Building Projects and Fixing Errors

With the files in place and CCS configurations set, the project is ready to be built. From the **project** menu, select **rebuild all**. Errors are reported and highlighted in red. Some errors likely relate to specifying the path of various included files; correct these paths to reflect your project and file locations. You can double click any red error line in the build window. This will open the suspect file and automatically position the cursor near the suspected error. Once you have corrected the errors, rebuild the project (actually, an incremental build is sufficient). Once successfully compiled, CCS generates an output file `lab4.out` that contains code for download to the DSP.

Before downloading the program, consider some details of the code. At this point, the program does not perform spectral inversion; you will perform that modification later. Instead, it is acting as a wire; output is equal to input. What is done in the main portion of `main.c`? What is done in `ISRs.c`? Look in the file `DSK_Support.c` for the subroutine called by the main code. Identify the interrupt number and source being used by the program. How does the program get into the interrupt routine?

5 Downloading and Running Programs

Connect an audio source to the DSP. Lab PC's have WINAMP available, which works well to source audio data. At some point, you will want to drive the system with the signal generators found in lab; set up the signal generator for this task. Be sure that whatever source that you use does not exceed about three volts

peak-to-peak. Split the DSP output to both an oscilloscope and the speakers on your PC. This will allow you to both see and hear the output of your system.

From the **file** menu, select **load program**. Choose your output file, `lab4.out`. Once the download is complete, select **debug** and **run**. If everything is operating correctly, you should hear and see output. Does the output appear to have any distortion? Determine a methodical way to investigate how the system responds to different inputs. For example, if you put a 100 Hz sinusoid on the input, what comes out? If you put a 100 Hz square wave on the input, what comes out? What happens if you change the frequencies of the sinusoid and square waves to 2kHz? Apply a 30kHz sinusoid to your DSK system. Listen (and describe) the output. Does the output make sense? Explain. What does this tell you about how the codec is configured on start-up? Fully document your observations.

Once you are finished with your tests, use CCS to halt the DSP.

6 Modifying Code to Perform Spectral Inversion

Spectral inversion is simply accomplished by $y[n] = (-1)^n x[n]$. Modify the C-code in `ISRs.c` to perform spectral inversion. Rebuild your project. Once all errors are eliminated, download the program to the DSP. Test your program and make sure it is working. For example, use the signal generator in lab to create a 1kHz sinusoid. Listen to this signal before applying it to the DSK. Next, apply the 1kHz signal to the DSK and listen to the output. What does it sound like? Does the output make sense?

Spectral inversion inverts sinusoids about the halfway point of the "folding frequency" of the system, which is one quarter the sampling rate of the codec. Use this fact to determine the current sampling rate of the codec. One nice way to do this accurately is to apply a variable-frequency sinusoid to the input. Connect both input and output to speakers. Adjust the input frequency until the two signals sound the same; the resulting frequency should be the system's "folding frequency".

Next, apply a speech signal and listen to the result. Can you understand the spectrally inverted speech? Humans hear in the 20Hz to 20kHz range, and many audio recordings admit this range of frequencies. Does this concern you for your application? Download and run your code (or team up with another student) on a second DSK. Apply an input signal to DSK1, run the output of DSK1 to the input of DSK2, and then connect the output of DSK2 to a speaker. What do you hear? Does it make sense?

7 Storing Samples for Analysis using CCS

CCS provides good data transfer and analysis capabilities, provided an adequate number of data points are available. Modify your code so that input and output are both length-128 vectors. Ensure that your code updates these buffers in a circular fashion. The modulus operator in C (the percent sign) may be tempting to use, but it is generally more efficient to use a logical-and as a mask. Recompile, download, and execute your code.

Next, open a watch window and view your input and output vectors. You will notice that the values will stay fixed unless you manually request that the data be refreshed. Although watch windows provide a means to view raw data, it is not convenient to view a lot of data. Thus, choose **view**, then **graph**, and then **time/frequency**. Under the start address, you can type the name of a variable or vector of interest, such as `input`. Set the acquisition and display buffer sizes to match the desired vector lengths. Additionally, set the data type to 16-bit signed integer. Experiment with the different display types, paying particular attention to the single time and FFT magnitude options. The display capabilities, while somewhat cumbersome, allow CCS to function as an oscilloscope and spectrum analyzer.

Determine a methodical way to use CCS data and display capabilities to investigate how the system responds to different inputs. For example, if you put a 1kHz sinusoid on the input, what comes out? If you put a 1kHz square wave on the input, what comes out? What happens if you change the frequencies of the sinusoid and square waves to 5kHz? Apply a 15kHz sinusoid to your DSK system. Listen (and describe) the output. Does the output make sense? Explain. Fully document your observations.

8 Reporting Requirements

To receive credit for this laboratory, you need to do a few things:

1. Complete all steps discussed in the laboratory handout.
2. Summarize your approach and findings in a concise, professional report. Be sure to answer all questions posed in the handout.
3. Attach a copy of your C-code programs as an appendix to your report.